

Exam 1 Review

15-110 – Monday 02/16

Announcements

- Check3 was due today
- Check2/Hw2 revision deadline **tomorrow (Tuesday) at noon!**
- No Gradescope exercise today (no new material)
- **Exam1 on Wednesday!**
 - Bring your paper notes (≤ 5 pages), something to write with, and your andrewID card
 - Arrive **early if possible** – we're checking IDs at the door

Announcements – Code Reviews

- **Code reviews!**

- **What:** meet with a TA for 10-15 minutes to get qualitative feedback on your code from your Hw2 submission. Attending the meeting and actively participating gets you 5 points on Hw3.
- **Why:** code style and structure are important, but not assessed by the autograder. The TA will point out different ways to solve the problems and areas where you can code more clearly or more robustly
 - Some students may be exempted from this meeting if they already have good style. We'll let you know if you're in that group by Monday EOD.
- **When:** this weekend (Friday-Sunday)
- **Where:** TA's choice

- **How to sign up for a code review slot**

- **Link:** <https://www.cs.cmu.edu/~110/hw/hw2-code-review.html>
- **Important:** sign-ups for each TA slot close 5pm Friday
- **Also important:** don't be late! If you are more than 3 minutes late to your meeting, you will not get credit on Hw3.
 - If something comes up and you need to cancel, notify the TA at least an hour before your timeslot. Do not do this multiple times.

Review Topics

- For Loops
- Looping over Strings
- Nesting

For Loops

For Loops

A **loop** is a control structure that lets you repeat a number of statements (the **body** of the loop) a certain number of times.

A **for-range loop** implements this looping by setting the loop control variable to a **pre-determined set of numbers**. The numbers are generated by the **range** expression.

We usually use for loops when we know **exactly how many times we need to loop**.

For Loop Structure

Recall that the basic structure of a for loop is:

```
for variableName in range(start, end, step):  
    LoopBody
```

Here's a simple example:

```
for num in range(1, 10): # step defaults to 1  
    print(num)
```

Example: Code Reading

Consider the following code snippet:

```
count = 0
for x in range(1, 101):
    if isPrime(x):
        print(x)
        count = count + 1
print("Total:", count)
```

If we assume that `isPrime` has been written and works correctly, what does this do?

Activity: sumTimesTable

You do: write a function `sumTimesTable(x)` that takes a number `x`, prints out the times table for `x` from `x * 1` to `x * 12`, and returns the sum of all the multiplied results. For example, `sumTimesTable(4)` would print:

4 * 1 = 4

4 * 2 = 8

4 * 3 = 12

...

4 * 12 = 48

and return **312** ($4+8+12+16+20+24+28+32+36+40+44+48=312$).

Looping Over Strings

Strings are Made of Characters

Strings are naturally composed of many individual parts – the individual characters that compose the string. This means we can write algorithms that work with the individual parts, using a **loop**.

How do we access a single character out of a string? Use indexing!

```
s = "testing"
```

```
i = 3
```

```
s[i] # "t"
```

Loop Over Indexes

How can we access **all** the characters in the string? Use a loop to visit each character, one at a time.

For now it's easiest to loop over the indexes of the string. Start at `0`, increment by `1`, and end at `len(s)`; we can use `range(len(s))` to visit all the positions.

```
for i in range(len(s)):
    print("Character:", s[i])
```

Problem Solving with Strings

What do we do inside a loop? It depends on the problem we're solving.

Consider any **data values** you'll want to keep track of and change over time. Store them in variables initially defined outside the loop.

Also consider when you'll know the answer and can **return** it (inside the loop? Outside?).

Example: countUpper

Problem: given a string, count the number of uppercase characters in the string.

```
def countUpper(s):  
    total = 0  
    for i in range(len(s)):  
        if s[i].isupper():  
            total = total + 1  
    return total
```

Activity: usesLetters

You do: the function `usesLetters(word, letters)` is given two strings, `word` and `letters`, and returns `True` if `word` only contains characters in the string `letters`, and `False` otherwise. For example:

```
usesLetters("happy", "ahpy") -> True
```

```
usesLetters("happy", "ahnry") -> False
```

```
usesLetters("aaaaaaa", "par") -> True
```

Nesting

Nesting Changes a Program's Control Flow

Nesting is the process of indenting control structures so that they occur inside other control structures. It is used to manipulate the control flow of a program to produce certain intended effects.

So far, we've learned about several control structures: **function definitions**, **conditionals**, **while loops**, and **for loops**. All of these structures have **bodies**, and each can be indented so it occurs inside the body of another structure.

Common Nested Structures - Functions

Though any nesting configuration you can think of is possible, some arrangements are more common than others.

Functions – we usually write function definitions at the top level of a program and nest conditionals/loops inside them when they're needed. When we **return** in a nested conditional/loop, we exit that structure and the whole function immediately.

```
def hasVowels(s):  
    for i in range(len(s)):  
        if s[i] in "aeiou":  
            return True  
    return False
```

Note how the loop is indented inside the function, and its body is indented again.

If the line `return True` is reached, the function will exit immediately without finishing the loop.

Common Nested Structures – Loop-Conditionals

Loop-Conditional – very often we nest a conditional inside a loop to check a certain property for every element that is iterated over.

While it's possible to pair an else with the nested if, it's only used if there's a clear alternative action. It's okay to do nothing on iterations that don't meet the requirement!

```
def countVowels(s):  
    result = 0  
    for i in range(len(s)):  
        if s[i] in "aeiou":  
            result = result + 1  
    return result
```

We don't need to update `result` if the letter isn't a vowel, so do nothing instead.

Common Nested Structures – Nested Loop

Nested Loop – if you need to iterate over multiple dimensions, a nested loop (one loop nested inside another) will manage the complex iteration. Each loop control variable manages one dimension.

It's important that the two loop control variables have different names so they can be referred to separately.

```
words = [ "testing", "one", "two", "three" ]  
for i in range(len(words)):  
    w = words[i]  
    for j in range(len(w)):  
        print(i, j, w[j])
```

The outer loop moves more 'slowly', as it only iterates once for each complete working of the inner loop.

Activity: trace code

You do:

What will the code to the right return when we call `mystery(6, 3)`?

Can we provide an input that makes it return ""?

```
def mystery(x, y):  
    s = ""  
    for i in range(x):  
        if i < y:  
            if i % 2 == 0:  
                s = s + "A"  
            else:  
                s = s + "B"  
            s = s + "C"  
        elif i == y:  
            j = 0  
            while j < i:  
                s = s + str(j)  
                j = j + 1  
        else:  
            s = s + "!"  
    return s
```